Jezik VHDL

Modeliranje sistemov

- Jezik VHDL je namenjen opisovanju digitalnih vezij
- Razlogi:
  - vedno bolj zahtevni sistemi,
  - dokumentacija,
  - testiranje z uporabo simulacije,
  - formalno preverjanje,
  - sinteza.

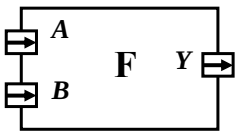
Modeliranje sistemov (nad.)

- Namen:
  - hitrejše načrtovanje vezij,
  - cenejše načrtovanje vezij,
  - zmanjšanje števila napak pri načrtovanju.

Opis stukture

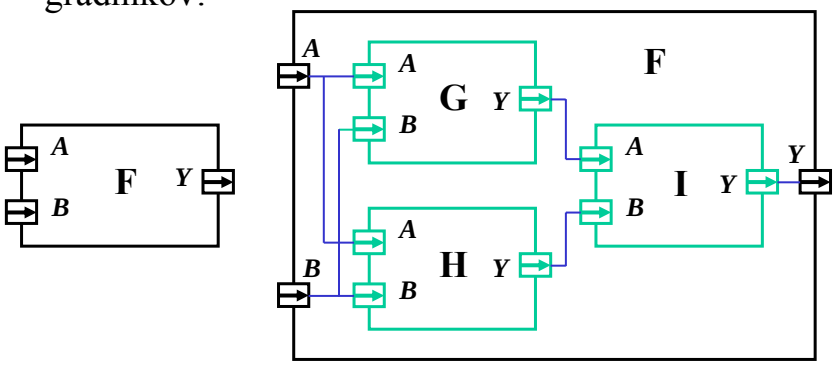
- Digitalni gradnik lahko opišemo:
  - z vhodi in izhodi,
  - z električnimi značilnostmi vhodov in izhodov.
- Primer VHDL gradnika (*entity*) z vhodoma *A* in *B* ter izhodom *Y*.

$$Y = F(A,B)$$



Opis stukture (nad.)

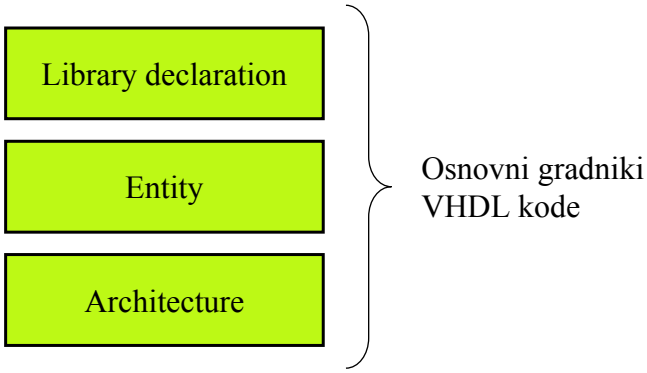
- Gradnik/enetiteto (*entity*) lahko sestavlja več gradnikov:



Opis stukture (nad.)

- Vsak gradnik, ki je del večjega gradnika je slika/primer (*instance*) osnovnega gradnika.
- Na prejšnji projekciji je predstavljen primer, kako je gradnik F sestavljen iz gradnikov G, H, I.
- Gradniki so med seboj povezani s signali.

Gradniki VHDL



Gradniki VHDL (nad.)

- Deklaracija entitete določa priključke entitete
- Telo entitete določa logično funkcijo entitete
- Deklaracija konfiguracije določa povezavo entitete z določeno arhitekturo
- Deklaracija knjižnice povezuje elemente
- Telo knjižnice vsebuje elemente iz deklaracije

Deklaracija entitete

```
library ieee;
use ieee.std_logic_1164.all ;

entity Adder is
  port ( A, B : in std_ulogic_vector(3 downto 0);
         Cin : in std_ulogic;
         Sum : out std_ulogic_vector(3 downto 0);
         Cout : out std_ulogic);
end Adder;
```

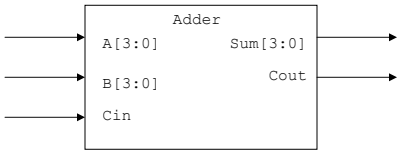
Podatki o knjižnici

Ime entitete

Ime priključka

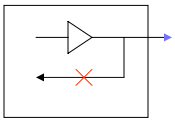
Način delovanja priključka

Tip priključka

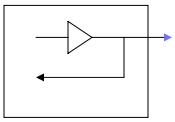


Način delovanja priključka

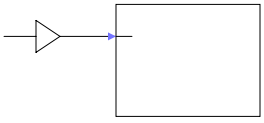
Način out



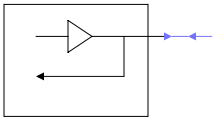
Način buffer



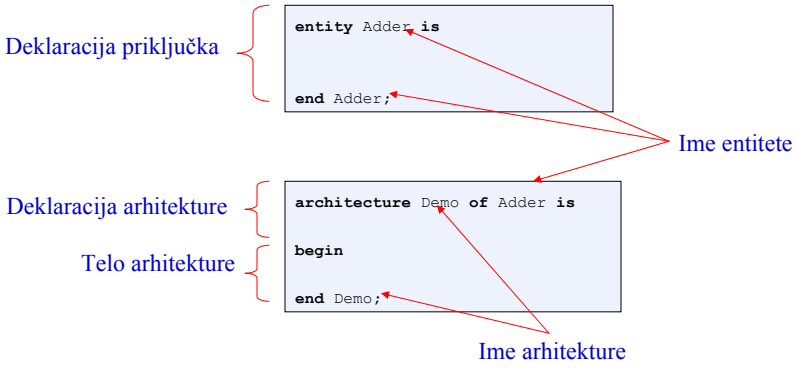
Način in



Način inout



Opis arhitekture

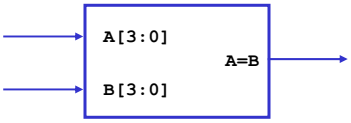


Način zapisa

- Logičen pogoj se lahko zapiše na tri načine:
  - zaporedno
  - paralelno
  - strukturalno

Primer primerjalnika

- Kot primer za načine zapisa si bomo ogledali primerjalnik:
  - vstopajo dva krat po štiri spremenljivke (A_0,A_1,A_2, A_3 in B_0, B_1, B_2, B_3),
  - izhod je rezultat primerjave (A=B).



Način zapisa (nad.)

- Ne glede na način zapisa ostaja skupni (navedba knjižnice in deklaracija entitete) del enak:

```
library ieee;
use ieee.std_logic_1164.all;

entity eqcomp4 is
  port  (A, B : in std_logic_vector(3 downto 0);
         Equals : out std_logic);
end eqcomp4;
```

Vhodi so deklarirani kot vektor

Izhod je samo eden

Zaporedni načina zapisa

- Primer zaporednega načina zapisa:
  - koda je sicer zaporedno napisana, vendar se moramo vedno zavedati, da se koda prevede v logično enačbo!

```
architecture seq1 of eqcomp4 is
begin
  process(A, B)
  begin
    if A = B then
      Equals <= '1';
    else
      Equals <= '0';
    end if;
  end process;
end seq1;
```


Zaporedni načina zapisa (nad.)

- Koda se ne izvaja zaporedno!!!
  - koda je sicer zaporedno napisana, vendar se moramo vedno zavedati, da se koda prevede v logično enačbo!

```
architecture seq2 of eqcomp4 is
begin
  process(A, B)
  begin
    Equals <= '0';
    if A = B then
      Equals <= '1';
    end if;
  end process;
end seq2;
```

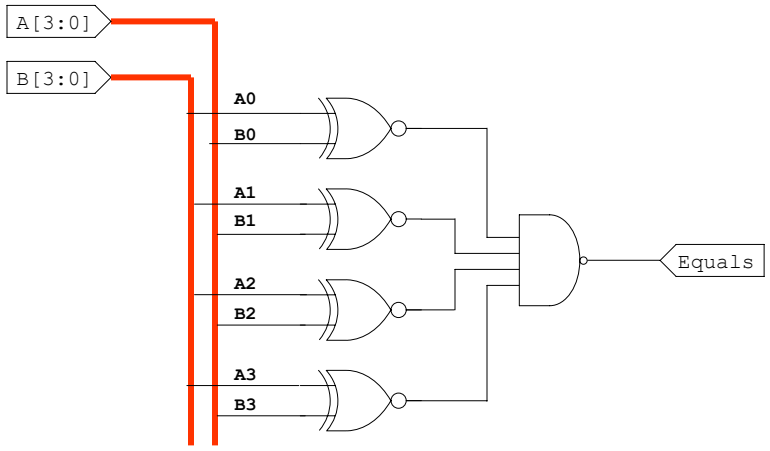
Paralelni način zapisa

- V tem načinu vezje opišemo z logičnimi enačbami:

```
architecture concurrent_bool of eqcomp4 is
begin
  Equals <= not(A(0) xor B(0)) and
            not(A(1) xor B(1)) and
            not(A(2) xor B(2)) and
            not(A(3) xor B(3));
end concurrent_bool;
```

```
architecture concurrent of eqcomp4 is
begin
  Equals <= '1' when (A = B) else '0';
end concurrent;
```


Strukturalni način zapisa



Strukturalni način zapisa (nad.)

- V tem načinu povezujemo gradnike med seboj.

Gradnik je lahko definiran posebej ali pa v knjižnici

```
architecture structure of eqcomp4 is
  signal X : std_logic_vector(3 downto 0);
begin
  u0 : xnor2 port map
    (A => A(0), B => B(0), O => X(0));
  u1 : xnor2 port map
    (A => A(1), B => B(1), O => X(1));
  u2 : xnor2 port map
    (A => A(2), B => B(2), O => X(2));
  u3 : xnor2 port map
    (A => A(3), B => B(3), O => X(3));
  u4 : and4 port map
    (A => X(0), B => X(2), C => X(3),
     D => X(4), O => Equals);
end structure;
```


Primer realizacije

- Primer realizacije preklopne funkcije:

  $Y = A\,B \vee C\,D$

- Realizacija konjunkcije in disjunkcije:

```
entity And2 is
port (A, B : in bit;
      Y : out bit);
end And2;

architecture Gate of And2 is
begin
  Y <= A and B;
end Gate;
```

```
entity Or2 is
port (A, B : in bit;
      Y : out bit);
end Or2;

architecture Gate of Or2 is
begin
  Y <= A or B;
end Gate;
```

```
entity AndOr is
  port ( A, B, C, D : in bit;
         Y : out bit);
end AndOr;

architecture struct of AndOr is
component Or2
    port ( A, B : in bit;
           Y : out bit);
  end component;

  component And2
    port ( A, B : in bit;
           Y : out bit);
  end component;

  signal A1, B1 : bit;

begin
  U1: And2 port map
    ( A => A, B => B, Y => A1);
  U2: And2 port map
    ( A => C, B => D, Y => B1);
  U3: Or2 port map
    ( A => A1, B => B1, Y => Y);
end struct;
```

Deklaracija entitete

Deklaracija uporabljenih entitet

Signali

Povezovanje uporabljenih entitet

Primer realizacije (nad.)

- Krajši način zapisa:

```
entity AndOr is
  port ( A, B, C, D : in bit;
         Y : out bit);
end AndOr;

architecture struct of AndOr is
  signal A1, B1 : bit;

begin
  U1: entity work.And2(gate)
    port map( A => A, B => B, Y => A1);
  U2: entity work.And2(gate)
    port map( A => C, B => D, Y => B1);
  U3: entity work.And2(gate)
    port map( A => A1, B => B1, Y => Y);
end struct;
```

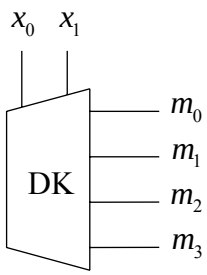
Primer realizacije (nad.)

- Enostavne logične operatorje lahko zapišemo v obliki enačbe:

```
entity AndOr is
  port ( A, B, C, D : in bit;
         Y : out bit);
end AndOr;

architecture struct of AndOr is
begin
  Y <= (A and B) or (C and D);
end struct;
```


Dekodirnik

- Primer dekodirnika

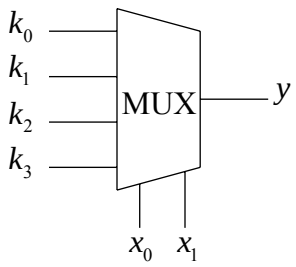


Dekodirnik (nad.)

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity Decoder is
  port (X: in std_logic_vector (1 downto 0);
        m: out std_logic_vector (3 downto 0);
end Decoder;

architecture struct of Decoder is
begin
   m <= "0001" when X="00" else
        "0010" when X="01" else
        "0100" when X="10" else
        "1000";
end struct;
```


Multiplekser

- Primer multiplekserja z dvema naslovnima vhodoma



Multiplekser (nad.)

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity mux_gate is
  port (X: in std_logic_vector (1 downto 0);
        K0,K1,K2,K3: in std_logic;
        Y: out std_logic);
end mux_gate;

architecture struct of mux_gate is

begin
   Y <= K0 when X="00" else
        K1 when X="01" else
        K2 when X="10" else
        K3;
end struct;
```


Postavitev generičnih tipov

- Generične tipe se uporablja pri postavitvi struktur, pri katerih se natančna velikost parametrov določi pri prenosu.
- Strukture so lahko postavljene neodvisno od velikosti parametrov
  - primer takšne strukture je lahko vsota dveh besed, pri čemer dolžina besede ni določena,
  - dolžina se pred implementacijo določi na višjem nivoju in se prenese preko parametrov

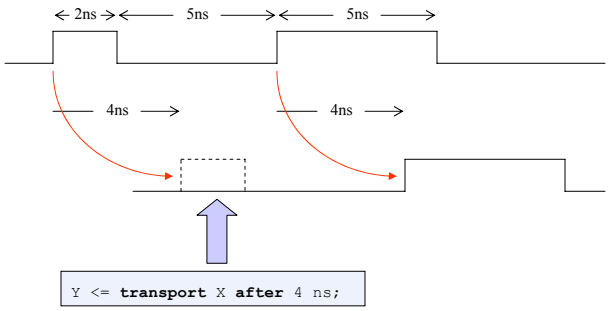
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Adder is
  generic (
    Width : integer range 2 to 32 := 16);
  port (
    A, B : in unsigned(Width-1 downto 0);
    Y : out unsigned(Width-1 downto 0));
  end Adder;

architecture RTL of Adder is
begin
  Y <= A + B;
end RTL
```

```
Adder_1: Adder
  generic map (Width => 22)
  port map (A => A, B => B, Y => Y);
```


Zakasnitve v vezjih

- Časovni parametri se uporabljajo samo za preverjanje pravilnosti delovanja v času simulacije,
- Signali, krajši od zakasnitve, se na izhodu ne pojavijo!
- Krajši signali se pojavijo samo pri uporabi ukaza "`transport`"

```
X <= '0', '1' after 1 ns, '0' after 3 ns, '1' after 8 ns, '0' after 13 ns;
Y <= X after 4 ns;
```



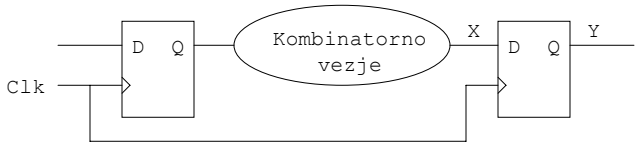

```
Y <= transport X after 4 ns;
```


Zakasnitve v vezjih (nad.)

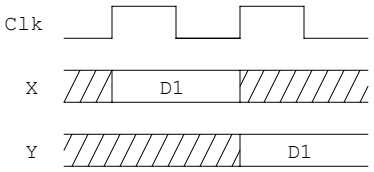
- Zakasnitev v vezjih ni možno določiti nad strukturami, ki se pred realizacijo v vezjih še spremenijo (minimizirajo)
- V spodnjem primeru zakasnitve ni možno izračunati:

```
Y <= ((A and B) or (C and D)) after 4 ns;
```

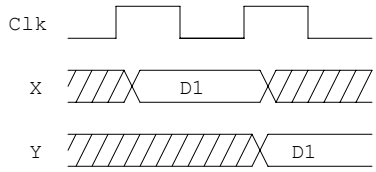
Model brez zakasnitve



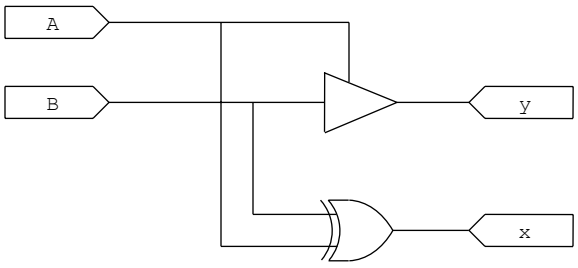
Brez zakasnitve



Z zakasnitvijo

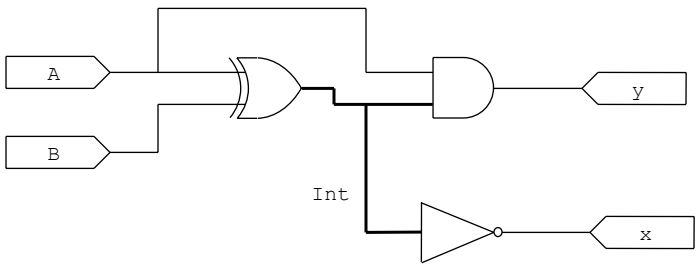


Paralelna izvedba



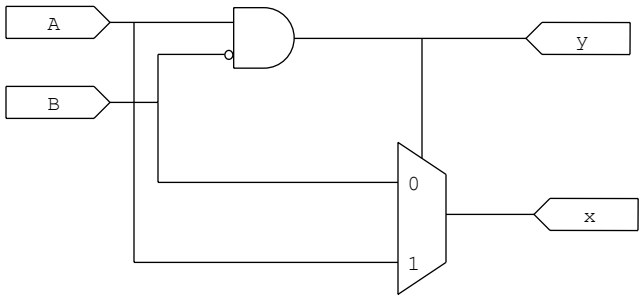
```
library ieee;
use ieee.std_logic_1164. all;

entity Test is
  port (A, B : in std_logic;
        X, Y : out std_logic);
end Test;

architecture Concurrent of Test is

begin

  X <= A xor B;

  with A select Y <= B when '1',
                     'Z' when '0',
                     '-' when others;

end Concurrent;
```


Notranji signali



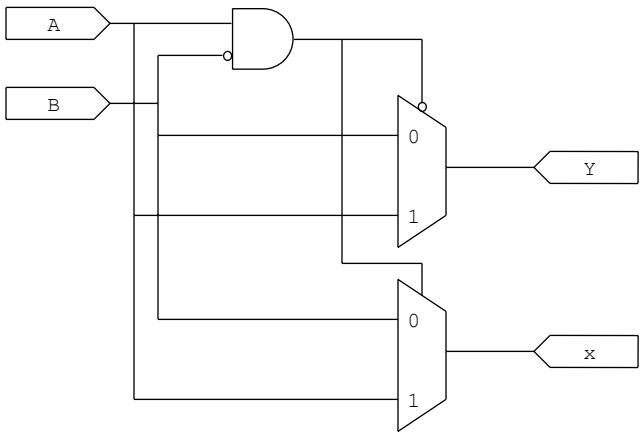
```
library ieee;
use ieee.std_logic_1164. all ;

entity Test is
  port (A, B : in std_logic;
  X, Y : out std_logic);
end Test;


architecture Internal of Test is

  signal Int : std_logic;

begin
  Int <= A xor B;
  X <= not Int ;
  Y <= Int and A;
end Internal;
```


Zaporedni stavki (procesi)



```
library ieee;
use ieee.std_logic_1164. all;

entity Test is
  port (A, B : in std_logic;
        X, Y : out std_logic);
end Test;

architecture Proc of Test is
begin
  P1: process (A, B)
  -- postavitev signalov ni dovoljena
   begin
    if A = '1' and B = '0' then
      X <= A;
      Y <= '1';
    else
      X <= B;
      Y <= '0';
    end if;
   end process P1;
 end Proc;
```


Zaporedni stavki (procesi)



```
library ieee;
use ieee.std_logic_1164.all;
entity Test is
  port(A, B : in std_logic;
       X, Y : out std_logic);
end Test;

architecture Proc of Test is
  signal Internal : std_logic;

begin
  P1 : process (A, B)
  begin
    if A = '1' and B = '0' then
      X <= A; Internal <= '0';
    else
      X <= B; Internal <= '1';
    end if;
  end process P1;
```

```
P2 : process (A, B, Internal)
  begin
    if Internal = '1' then
      Y <= A;
    else
      Y <= B;
    end if;
  end process P2;

end Proc;
```

Procesa se med simulacijo izvajata paralelno!

Primerjava zaporednih in paralelnih stavkov

```
architecture Concurrent of Test is
begin
  Y <= A or B;
  Y <= C and D;
end Concurrent;
```

Prevajalnik javi napako

```
architecture Sequential of Test is
begin
  process(A, B, C, D)
  begin
    Y <= A or B;
    Y <= C and D;
  end process;
end Sequential;
```

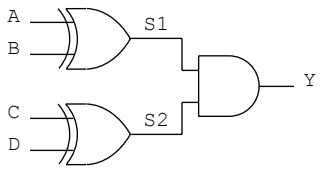
Prevajalnik upošteva zadnji izraz

Simulacija

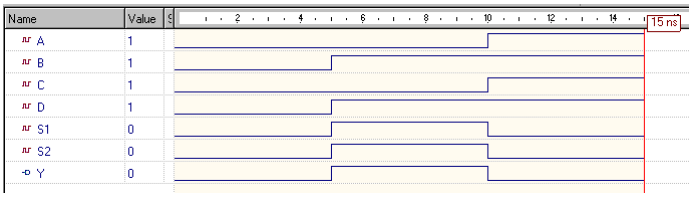
- Simulacija se izvaja na osnovi dogodkov (sprememb vrednosti signalov).
- Pri vsaki spremembi signala se izračuna nova vrednost preklopne funkcije.
- Izračun se ponavlja dokler ni več nobene spremembe signalov v vezju.
- Ko ni več sprememb, se realen čas pomakne na naslednjo spremembo na vhodu.

Primer simulacije

```
architecture sim of Test is
  signal A, B, C, D : std_logic := '0';
  signal S1, S2, Y : std_logic;

begin
  A <= '0' after 5 ns, '1' after 10 ns;
  B <= '1' after 5 ns;
  C <= '0' after 5 ns, '1' after 10 ns;
  D <= '1' after 5 ns;
  S1 <= A xor B;
  S2 <= C xor D;
  Y <= S1 and S2;
end Sim;
```



t(ns)	Δ	A	B	C	D	S1	S2	Y
0	+0	0	0	0	0	U	U	U
0	+1	0	0	0	0	0	0	U
0	+2	0	0	0	0	0	0	0
5	+0	0	1	0	1	0	0	0
5	+1	0	1	0	1	1	1	0
5	+2	0	1	0	1	1	1	1
10	+0	1	1	1	1	1	1	1
10	+1	1	1	1	1	0	0	1
10	+2	1	1	1	1	0	0	0



Definicija konstant

- Poljubnemu podatkovnemu tipu lahko priredimo ustrezno konstantno vrednost.
- Primer postavitve konstant:

```
constant Mult : std_logic_vector := "0001"; -- Opcode multiply
constant Width : integer := 12;
```

Signali

- Signale si lahko predstavljamo kot povezave med elementi.
- Signali lahko predstavljajo tudi pomnilne celice.

```
signal shiftReg : std_logic_vector(7 downto 0);
shiftreg <= shiftreg(6 downto 0) & Input;
```

- Signalom se lahko priredi začetna vrednost, vendar ta običajno ne vpliva na sintezo vezja.

```
signal Count : std_logic_vector(3 downto 0) := "0101";
```


Spremenljivke

- Spremenljivke se uporabljajo samo v procesih in podprogramih.

```
architecture will_work of my_and is
begin
adding: process (a_bus)
    variable tmp: bit;
  begin
    tmp := '1';
    for i in 7 downto 0 loop
      tmp := a_bus(i) and tmp;
    end loop;
    x <= tmp;
  end process;
end will_work;
```

Primeri podatkovnih tipov

Števni tipi

```
type MyBit is ('0', '1');
type Barva is (Zelena, Rumena, Modra, Bela);
```

Celoštevilčni tipi – privzeta dolžina je 32 bitov

```
type Count is integer range 0 to 10;
```

Polja

```
type MyBitVector is array (natural range <>) of Mybit;
type MyByte is array (natural range 7 downto 0) of Mybit;
```


Zapis

```
type FloatType is record
                    Sign : MyBit;
                    Mantissa : MyBitVector(7 downto 0);
                    Exponent : MyBitVector(15 downto 0);
                  end record;
```

Podtipi

```
subtype Byte is std_ulogic_vector 7 downto 0;
```

Primeri standardnih tipov

```
package STANDARD is
  type boolean is (false, true);
  type bit is ('0', '1');
  type character is ( ASCII chars... );
  type severity_level is (note, warning, error, failure);
  type integer is range -2147483648 to 2147483647;
  type real is range -1.0E308 to 1.0E308;
  type time is range -2147483647 to 2147483647
  units
    fs;
    ps = 1000 fs;
    ns = 1000 ps;
    us = 1000 ns;
    ms = 1000 us;
    sec = 1000 ms;
    min = 60 sec;
    hr = 60 min;
  end units;
  ...
```



```
  ...
  subtype delay_length is time range 0 fs to time'high;
  impure function now return delay_length;
  subtype natural is integer range 0 to integer'high;
  subtype positive is integer range 1 to integer'high;
  type string is array (positive range <>) of character;
  type bit_vector is array (natural range <>) of bit;
  type file_open_kind is (
    read_mode,
    write_mode,
    append_mode);
  type file_open_status is (
    open_ok,
    status_error,
    name_error,
    mode_error);
  attribute foreign : string;
end STANDARD;
```

Atributi

- Primer atributov, ki so definirani v jeziku VHDL:

```
clock'event vrne true pri spremembi vrednosti ure

signal A : unsigned(3 downto 0)

A'left vrne 3
A'right vrne 0
A'range vrne 3 downto 0
A'length vrne 4
```


Tipi paketa std_logic_1164

Tip `bit` je definiran:

```
type bit is ('0', '1');
```

Tip `std_ulogic` v paketu `std_logic_1164` je definiran:

```
type std_ulogic is ( 'U', -- Uninitialized
                     'X', -- Forcing Unknown
                     '0', -- Forcing 0
                     '1', -- Forcing 1
                     'Z', -- High Impedance
                     'W', -- Weak Unknown
                     'L', -- Weak 0
                     'H', -- Weak 1
                     '-' -- Don't care
                   );
```

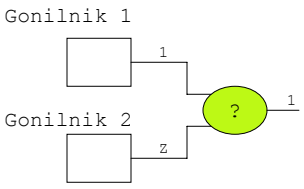
Vektor `std_ulogic_vector` je definiran:

```
type std_ulogic_vector is array ( natural range <> ) of std_ulogic;
```

Pravilnostna tabela std_logic

- Pri povezanih izhodih, je potrebno upoštevati razširjeno pravilnostno tabelo.

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X



Funkcije paketa std_logic_1164

Logične funkcije:
and, nand, or, xor, xnor, not

Funkcije za pretvorbo tipov:
To_bit, To_bitvector, To_StdULogic,
To_StdULogicVector, ToStdLogicVector

Funkcije za določanje fronte:
rising_edge() and falling_edge()

Paket std_logic_1164 ne vsebuje aritmetičnih funkcij!

Paket numeric_std

- V paketu numeric_std so postavljeni numerični tipi:

```
type unsigned is array (natural range <>) of std_logic;
type signed is array (natural range <>) of std_logic;
```

- V paketu numeric_std so postavljene numerične funkcije in funkcije za pretvorbo tipov.

Primeri pretvorbe

```
architecture Y of X is
  signal S : std_logic_vector(7 downto 0);
  signal A, B : unsigned(7 downto 0);
  signal MyInt : integer;
begin
  S <= B;  -- prevajalnik javi napako

  S <= std_logic_vector(A);     -- pravilna pretvorba tipa

  S <= std_logic_vector(A + B); -- vsota je definirana v paketu

  MyInt <= to_integer(A + B);   -- pretvorba v integer

  A <= unsigned(S) + MyInt;     -- napačni tipi
end Y;
```

Prirejanje poljem

```
architecture assign of Examples is
  signal Byte : std_logic_vector(7 downto 0) := "01010101";
  signal Word : std_logic_vector(15 downto 0);

begin
  Byte <= "00001111";
  Byte <= ('1', '0', '1', '0', '1', '0', '1', '0');
  Byte <= X"0F";
  Byte <= (others => '1');
  Byte <= (7 | 6 => '1', others => '0');
  Byte <= (7 => '1', 4 => '1', 3 downto 1 => '0', others => '0');
  Word <= X"FF" & Byte;
  Word(15 downto 8) <= (others => Byte(7));
  Word(7 downto 0) <= Byte;
end Examples;
```


Stavek *wait*

```
process - Tvorba signala za reset
begin
  Reset <= '0';
  wait for 15 ns;
  Reset <= '1';
  wait;
end process;
```

```
process  -- Tvorba kratkega impulza
begin
  Spare_n <= '1';
  wait for 200 ns;
  Spare_n <= '0';
  wait for 8 ms;
  Spare_n <= '1';
  wait;
end process;
```

```
process  -- Tvorba urinega impulza
begin
  Clk <= '0';
  wait for 10 ns;
  Clk <= '1';
  wait for 10 ns;
end process;
```

```
process -- Čakaj, da se urin impulz spremeni na 0
begin
  wait until Clk = '0';
  A <= Stimulus;
end process
```

```
process -- Čakaj na spremembo signalov A, B ali C
begin
  wait on A, B, C;
  X <= A and b and C;
end process;
```


Stavek *if*

```
if Count = 1 then
  OutPut <= '1';
else
  OutPut <= '0';
end if;
```

```
if Count = 1 then
  A := 10;
elsif State = Idle then
  A := 20;
else
  A := 30;
end if;
```

Stavek *case*

```
case PresentState is
  when Idle =>
    Output <= '1';
    NexState <= S1;
  when S1 =>
    Output <= '0';
    NexState <= S2;
  when S2 =>
    Output <= '1';
    NexState <= S1;
end case;
```



```
case OPCode is
  when "0000" | "0001" | "0010" | "0011" =>
    Result <= A;
  when "1000" | "1001" | "1010" | "1011" | "1100" | "1101" | "1110" | "1111" =>
    Result <= B;
  when "0100" | "0101" =>
    Result <= C;
  when others => -- All possible cases must be covered
    null; -- do nothing
end case;
```

Stavki *loop, next, exit*

```
A := 0;
  for i in 0 to 10 loop
    A := A + B(i);
end loop;
```

```
for i in 0 to 10 loop
  next when i = 1;  -- preskoči ukaze pri izpolnjenem pogoju
  A := A + B(i);
end loop;
```

```
while A > B loop
  B := B * 2;
end loop;
```



```
i := 0;
loop
  i := i + 1;
  exit when i = 10;
end loop;
```

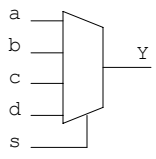
```
L1 : for i in 10 downto 0 loop
  L2 : loop
    next L1 when Done = '1';  -- pri izpolnjenem pogoju L1
  end loop L2;
end loop L1;
```

Stavki *with, select, when*

```
with selection_signal select
  signal_name <= value_a when value_1_of_selection_signal,
                 value_b when value_2_of_selection_signal,
                 value_c when value_3_of_selection_signal,
                 value_x when last_value_of_selection_signal;


-- če niso naštete vse možnosti, mora biti zadnja others
```


Stavka *when, else*

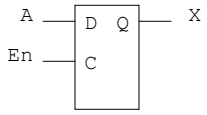
```
signal_name <= value_a when condition_1 else,
               value_b when condition_2 else,
               value_c when condition_3 else,
               value_x;
```




```
Y <= a when (s = "00") else,
     b when (s = "01") else,
     c when (s = "10") else,
     d;
```

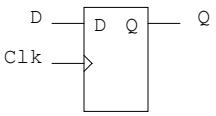
Pomnilna celica - *latch*

```
library ieee;
use ieee.std_logic_1164. all;

entity Latches is
  port (
  A, En : in std_logic;
  X  : out std_logic);
end Latches;

architecture RTL of Latches is
  begin
    P1 : process(En, A)
    begin
      if En = '1' then
        X <= A;
    end if;
  end process;
end RTL;
```



D- pomnilna celica

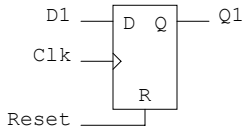
```
library ieee;
use ieee.std_logic_1164. all;

entity Latches is
  port (
  D, Clk : in std_logic;
  Q  : out std_logic);
end Latches;

architecture RTL of Latches is
  begin
    P1 : process(Clk)
    begin
      if Clk = '1' and Clk'event then
        Q <= D;
    end if;
  end process;
end RTL;
```

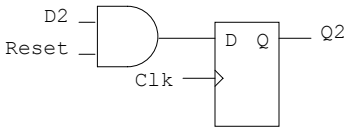


D- pomnilna celica z vhodom *reset*

```
ASynch : process(Clk, Reset)
begin
  if Reset = '1' then
    Q1 <= '0';
  elsif rising_edge(Clk) then
    Q1 <= D1;
  end if;
end process;
```

```
Synch : process(Clk)
begin
  if rising_edge(Clk) then
    if Reset = '1' then
      Q2 <= '0';
    else
      Q2 <= D2;
    end if;
  end if;
end process;
```





8-bitni števec

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity Counter is
  port(Clk, Reset, Enable : in std_logic;
       Count : out unsigned(7 downto 0));
end Counter;
architecture RTL of Counter is
begin
  Cnt : process(Clk, Reset)
    variable CountReg : unsigned(7 downto 0);
  begin
    if Reset = '1' then
      CountReg := (others => '0');
    elsif rising_edge(Clk) then
      if Enable = '1' then
        CountReg := CountReg + 1;
      end if;
    end if;
    Count <= CountReg;
  end process Cnt;
end RTL;
```

Procedure in funkcije

- Funkcija ima, za razliko od procedure, samo en izhod.
- Pri postavljanju funkcij in procedur je potrebno upoštevati, da se bodo te prevedla v obliko, ki bo primerna za zapis v integrirano vezje.
- Postavijo se lahko različne funkcije z enakim imenom, ki pa se razlikujejo po tipu parametrov *(Overloading operators)*.


```
library ieee;
use ieee.std_logic_1164.all;

package smd098_pkg is
  function OR_reduce (A : std_logic_vector)
                      return std_logic;
end smd098_pkg;

package body smd098_pkg is
  function OR_reduce (A : std_logic_vector)
                      return std_logic is

  variable temp : std_logic;

  begin
    temp := '0';
    L1: for i in A'range loop
      temp := temp or A(i);
    end loop L1;
    return temp;
  end function OR_reduce;
end smd098_pkg;
```

```
library ieee;
use ieee.std_logic_1164.all;
use work.smd098_pkg.all;

entity test_or is
  port (A : in std_logic_vector(3 downto 0);
        Y1, Y2 : out std_logic);
end test_or;

architecture RTL of test_or is
begin
  -- Concurrent fun. call
  Y1 <= OR_reduce(A);
  process(A)
  begin
    -- Sequential fun. call
    Y2 <= OR_reduce(A);
  end process;
end RTL;
```



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package smd098_pkg is
  function OR_reduce (A : std_logic_vector) return std_logic;
  function OR_reduce (A : unsigned) return std_logic;
end smd098_pkg;

package body smd098_pkg is
  function OR_reduce (A : std_logic_vector) return std_logic is
  variable temp : std_logic;
   begin
    temp := '0';
    L1 : for i in A'range loop
      temp := temp or A(i);
    end loop L1;
    return temp;
  end function OR_reduce;

  function OR_reduce (A : unsigned) return std_logic is
  variable temp : std_logic;
  begin
    temp := '0';
    L1 : for i in A'range loop
      temp := temp or A(i);
    end loop L1;
    return temp;
  end function OR_reduce;
end smd098_pkg;
```

```
package smd098_pkg is

  procedure DFF (signal Clk : in std_logic;
                 signal D : in std_logic_vector;
                 signal Q : out std_logic_vector);
end smd098_pkg;

package body smd098_pkg is
  procedure DFF (signal Clk : in std_logic;
                 signal D : in std_logic_vector;
                 signal Q : out std_logic_vector) is
  begin
    if rising_edge(Clk) then
      Q <= D;
    end if;
  end procedure DFF;
end smd098_pkg;
```



```
library ieee;
use ieee.std_logic_1164.all;
use work.smd098_pkg.all;

entity test_DFF is
  port (Clk : in std_logic;
        A : in std_logic_vector(3 downto 0);
        Y1, Y2, Y3 : out std_logic_vector(3 downto 0));
end test_DFF;

architecture RTL of test_DFF is
  begin
    -- Concurrent proc. call
    DFF(Clk => Clk, D => A, Q => Y1);
    -- Concurrent proc. call, positional association
    DFF(Clk, A, Y2);
    -- Sequential procedure call
    process(Clk)
    begin
      DFF(Clk, A, Y3);
    end process;
end RTL;
```

Knjižnice

- V knjižnicah lahko postavimo:
  - definicije tipov,
  - deklaracija konstant,
  - funkcije in procedure,
  - komponente.
- Primer vključevanja knjižnice:

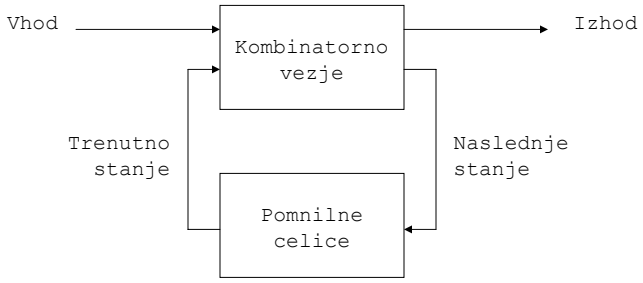
```
library mylib;
use mylib.mypackage.all;
```

```
library mylib;
use mylib.mypackage.myfunc;
```

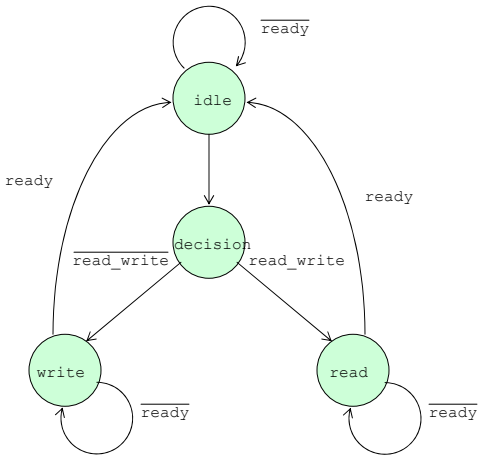
Knjižnica Zbirka Element

Sekvenčna vezja

- Sekvenčno vezje je sestavljeno iz kombinatornega dela in pomnilnih celic:



Primer avtomata



Izhodna funkcija

State	oe	we
idle	0	0
decision	0	0
write	0	1
read	1	0


```
entity example is port (
          read_write, ready, clk  : in bit;
          oe, we                  : out bit);
end example;

architecture state_machine of example is
          type StateType is (idle, decision, read, write);
          signal present_state, next_state : StateType;
begin
state_comb:process(present_state, read_write, ready) begin
          case present_state is
                    when idle =>                  oe <= '0'; we <= '0';
                              if ready = '1' then
                                        next_state <= decision;
                              else
                                        next_state <= idle;
                              end if;
                    when decision =>              oe <= '0'; we <= '0';
                              if (read_write = '1') then
                                        next_state <= read;
                              else        --read_write='0'
                                        next_state <= write;
                              end if;

...
```

```
...

                    when read =>                            oe <= '1'; we <= '0';
                              if (ready = '1') then
                                        next_state <= idle;
                              else
                                        next_state <= read;
                              end if;
                    when write =>                           oe <= '0'; we <= '1';
                              if (ready = '1') then
                                        next_state <= idle;
                              else
                                        next_state <= write;
                              end if;
          end case;
end process state_comb;

state_clocked:process(clk) begin
          if (clk'event and clk='1') then
                    present_state <= next_state;
          end if;
end process state_clocked;

end architecture state_machine; --"architecture" is optional; for clarity
```